AD-A278 680

An Architecture for A Synthetic Vacuum Cleaner

DTIC
ELECTE
APR 2 9 1994
S D
F

R. James Firby*
Computer Science Department
University of Chicago
1100 East 58th Street, Chicago, IL 60637
firby@cs.uchicago.edu

Abstract

This paper discusses three increasingly complex definitions of the vacuum cleaning task. Each definition requires different capabilities in the vacuuming agent and hence different internal software architectures. However, the definitions suggested form a progression from reactive, to synthetic, to intelligent, and the lessons learned looking at one problem supply important insights for tackling the next. The paper then describes the Animate Agent project which defines an architecture for robot control that attempts to embody both the reactive methods of control required in simple vacuum cleaners and the symbolic methods of situation classification and plan choice required by synthetic vacuum cleaners.

To appear in the Proceedings of the AAAI Fall Symposium on Instantiating Real-World Agents

This document has been approved for public release and sale; its distribution is unlimited

1 Introduction

Vacuuming a room autonomously without engineering the agent/task too much is an interesting focus for discussion because it touches on a number of issues that have received scant attention in the current AI literature. The task requires both symbolic and continuous control and the use of sophisticated perception. In fact, to do the job "right" also requires natural language understanding tightly integrated with the state of the vacuum cleaning task as it is being performed.

1.1 Defining the Problem

Vacuum cleaning is an excellent example of a real world problem that can be made arbitrarily simple, or arbitrarily complex, depending on exactly how the task is defined. For example, the following questions all seem like legitimate potential elaborations to the basic scenario of vacuuming a room that is empty except for a few, immovable obstacles:

- Will the room contain furniture with complex shapes and many floor-level nooks and crannies?
- Will the floor be cluttered with objects that should be moved or picked up? Chairs or magazines for instance.
- Will children or other moving agents be present? Will they be cooperative or should they be ignored?
- Should the vacuum be responsive to long term user strategies? For example, "Vacuum the living room every morning but vacuum the bedroom only Saturday when everyone has left the house."
- Should the vacuum be responsive to short term user requests? "Don't do that now, come back later," or "Do that side of the room first today."

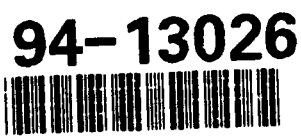
An appropriate solution to the autonomous vacuum cleaning problem is determined by the details of the task specification. The software architecture, knowledge representation, and programming methodology used for one task may not work well for another.

It is important for this workshop that we define the problem carefully. We must realize that if we define the problem to be a simple reactive one, then more complex architecture will be overkill and appear irrelevant. We must allow enough complexity into the problem to let these architectures make their point, if they can.

This paper will discuss three increasingly complex definitions of the vacuum cleaning task. Each definition requires different capabilities in the vacuuming agent and hence different internal software archi-

*This work was supported in part by ONR grant N00014-93-1-0332.

94-13026



DTIC QUALITY INSPECTED 3

94 4 28 12 8

Best
Available
Copy

tectures. However, the definitions suggested form a progression and the lessons learned looking at one problem supply important insights for tackling the next. For example, "reactive" robot architectures have laid important groundwork for the "symbolic" robot architectures that must come next.

We should be searching for a progression of problems that makes sense.

After defining the problem, the paper describes the basic ideas behind the Animate Agent project. This project defines an architecture for robot control that attempts to embody both the reactive methods of control required in simple vacuum cleaners and the symbolic methods of situation classification and plan choice required in synthetic vacuum cleaners.

2 A Vacuum for Every Task

The following three tasks are proposed as a way to organize discussion of the vacuum problem. Each task builds on the one before but each requires new programming and knowledge representation techniques to cope with expanded aspects of the problem.

2.1 The "Simple" Vacuum

A central aspect of the vacuum cleaning problem is that the vacuum cleaner must move. This brings up a host of immediate issues that can alter the problem considerably. First, there is the question of whether objects in the room to be cleaned are fixed or can move. Most real furniture moves around from time to time and a vacuum cleaner should be able to cope. Similarly, there may be transient objects that are only in the room sometimes and effectively look like objects that have moved. In either case, any *a priori* knowledge the vacuum may have will necessarily be unreliable.

A second issue is whether or not there will be objects moving in the room while vacuuming is under way. Realistically, a vacuum cleaner will almost always encounter moving objects on occasion because someone is bound to enter the room even when they are forbidden to do so. Thus, a vacuum cleaner must have a strategy for coping with moving objects even if it is as simple as stopping or trying to go around.

Third, is the question of whether the objects in the room to be vacuumed are complex or simple with respect to the vacuum cleaner. Simple objects can be vacuumed around by tracing their perimeter (or some other "simple" strategy) while complex objects may require intricate steering maneuvers to ensure that nooks and crannies are cleaned effectively. Increasing the complexity of objects can have a significant effect on the kind of software architecture required to steer the robot.

Let us define the simple vacuum problem as the task of cleaning rooms that hold simple objects. We will also assume that the objects move around somewhat between cleaning and that people occasionally move through the room. There is no simpler problem worth considering at this symposium.

2.1.1 Using Reactive Strategies

The simple vacuum task is a natural candidate for the use of reactive approaches to robot control. Typically, such solutions would use very little state and simple sensing strategies to differentiate rug from non-rug. The resulting vacuum would embody some strategy for covering the entire room, such as a random walk or a slow spiral outward that relies only on local sensing. It would be able to avoid non-rug areas (perhaps differentiating between non-rug floor and actual obstacles) and it may repeat the vacuuming procedure some number of times to account for moving obstacles that could have caused it to miss parts of the rug earlier. Such strategies might be implemented using subsumption [2], GAPPS and REX [8], ALFA [5], or any number of other languages that map the current state into actions through a decision network or collection of concurrent processes.

A reactive approach to vacuuming is attractive because the simple vacuuming task contains a good deal of uncertainty, unpredictability, and lack of knowledge. These things are exactly what reactive approaches to robot control are designed to cope with. The driving force behind the reactive idea is the need to deal quickly and effectively with a changing and uncertain environment. The idea works well because a variety of real world tasks are easily achieved using simple reactive strategies that require only immediate local sensory data. The simple vacuum problem is just such a task.

2.2 The "Synthetic" Vacuum

A more sophisticated autonomous vacuum would be able to apply different vacuuming strategies at different times in different situations. Differentiating situations that require alternative strategies will often depend on sensor information, experience, instructions, and vacuuming knowledge.

For example, consider the need to vacuum around and under complex furniture. One approach is to examine the piece of furniture (or perhaps just the spaces needing cleaning) and retrieve or create a plan to vacuum that area effectively.

A similar situation arises when the floor can be littered with objects that need to be picked up and put away, or at moved and vacuumed underneath. A natural solution is to classify each object and choose a plan to move it appropriately. One might also want the vacuum to adopt different vacuuming strategies

Codes
Dist | Avail and / or Special
A-1

when different types of object are moving around the room. Perhaps adults can be safely ignored but when a child enters the room the vacuum should stop and wait for the child to leave.

The vacuum cleaner may also need to carry out special instructions from its user. For example, the user may want the vacuum to follow different plans in different situations: vacuuming the living room only when no one is home, or vacuuming the west side of the room first and then the east. Even if we ignore the problem of how the user specifies that information, the robot still has to be able to tell situations apart and apply different plans.

Let us define the synthetic vacuum problem as the task of cleaning rooms that contain complex furniture and moving and stationary objects that have to be treated in different ways. Furthermore, the vacuum should be capable of following a variety of user instructions in different situations.

2.2.1 Symbols and Representations

Solving the synthetic vacuum problem requires gathering information, classifying the situation, and choosing a plan to apply. The vacuum might adopt a strategy for methodically vacuuming the local region of carpet and mapping its extent. Equipped with the map, the system could keep track of those regions of carpet it had completed and those areas of the room not yet explored. It might also attempt to classify obstacles and non-rug areas as pieces of furniture that cannot be moved, those that can be moved, items on the floor that should be put somewhere else, or objects (like children and pets) that are likely to move on their own. The vacuum would make and use plans for exploring and cleaning areas of carpet as they were found and for dealing with the objects it encountered along the way. Such a system would have a good idea of when it was done because it could tell by its map and its object classifications. A variety of architectures for managing plans dynamically would be appropriate for such a system: RAPs [4], PRS [6], and Universal Plans [11], to name just a few.

Situations and plans are the natural vocabulary for discussing solutions to the synthetic vacuum task because we, as humans, seem to conceptualize the problem in those terms. We can recognize and classify objects and situations with apparent ease and we communicate knowledge of how to deal with different situations in terms of prescribed actions. In effect, we divide the world into symbols and our activities into discrete actions. The synthetic vacuum task fits this mold.

2.2.2 Using Symbols and Reactivity

The primary problem with the use of robot architectures that depend on symbolic classifications and discrete plans of action is that they often have trouble remaining "reactive." While it seems natural to classify the world into states and select different plans in different states, none of the simple vacuum problems have gone away. The robot must continue to assume that unpredictability and lack of knowledge will abound and it must continue to use reactive techniques for actually taking action in the world.

Strategies for local navigation, obstacle avoidance, and sofa or wall-following are much more easily thought of and coded up as continuous processes (or behaviors, routines, or skills). Symbolic systems have a notoriously hard time with such problems. On the other hand, mapping, object classification (and memory) and the selection of different strategies for coping with different objects in different situations are often more productively thought of and coded up as symbolic programs. Once a reactive system starts to build a map, choose among object recognition strategies, and select control actions based on both the external state and its internal map, it becomes harder and harder to describe in terms of concurrent continuous processes.

Thus, the synthetic vacuum problem does not stand on its own, it is an extension of the simple vacuum task. No solution to the synthetic problem can ignore the reactive requirements of the simple task. Software architectures for real vacuums must include and coordinate both symbolic plans and continuous reactive processes.

2.3 The "Intelligent" Vacuum

The vacuum cleaning problem can be extended further to include the ability to negotiate with its user and understand and respond in a reasonable way to commands like "Stop that," "Vacuum here later," "Do under the sofa first" or "Stay away from the baby." The system must be able to understand its own actions at multiple levels of abstraction to understand and respond to such requests properly. Such commands should also continue to influence the system's behavior for some time into the future.

We will define the intelligent vacuum problem as the task of cleaning the carpet in any reasonable situation and responding to user input (or any other stimulus) in a reasonable way.

2.3.1 Using Intelligence

The intelligent vacuum builds on both the synthetic and simple vacuuming tasks. An intelligent vacuum needs to cope with the real world reactively, represent plans of action, perceive and clas-

sify objects and situations, and understand its plans and actions well enough to alter them appropriately while conversing with a user.

I believe that discussing vacuuming problems that are less complex than the simple task is pointless. The simple vacuum requires the minimum capabilities demanded by the real world. I also believe that the simple vacuum will find little application in the real world. It just isn't responsive enough to user requirements. A synthetic vacuum is probably necessary for industrial applications and nothing short of the intelligent vacuum will do in the home.

3 The Animate Agent Architecture

The Animate Agent Project is part of an on-going effort at the University of Chicago to understand the mechanisms necessary to create intelligent, goal-directed behavior in software and hardware agents. Current work is specifically aimed at building the type of system capable of performing the synthetic vacuum task: a system that embodies both reactive control and symbolic choice of discrete plans.

Three major observations about the world drive this project:

1. An agent will not know or control everything about the world in which it must act. As a result, details about the future will be impossible to derive and the agent will not be able to construct detailed plans for achieving its goals in advance of their execution.
2. An agent will typically find itself in situations that are not significantly different from those it has encountered in the past. Therefore, it makes sense for the agent to maintain a library of specific methods for doing things in specific situations.
3. An agent will often use extremely high bandwidth sensors to observe the world (*i.e.*, vision). Processing the data from such sensors will tax the real-time computational power of the agent and it will have to resort to different special purpose processing algorithms in different situations.

The overall system design for the project is shown in Figure 1 and consists of three components: a planner, a reactive plan executor, and a control system. The planner takes the agent's goals and breaks them down into steps that can be more readily executed (it i.e.a plan). However, because the planner's knowledge of the world, and hence the future, will always be uncertain, it can only produce *sketchy plans* that leave many steps to be chosen as the actual situation unfolds. The executor makes these choices at run time when the true state of the world can be sensed. The output from the executor is a detailed set of instructions for configuring the control system to implement reactive skills (after [13]) that make use of the agent's sensors and effectors.

3.1 The RAP Execution System

The RAP execution system was designed and implemented by one of us (Firby) and is described in [4]. The RAP system expands vague plan steps into detailed instructions at run time by choosing an appropriate method for the next step from a preexisting library. By waiting until run time, knowledge of the situation will be direct rather than predicted and much of the uncertainty and incompleteness plaguing the planner will have disappeared. However, some uncertainty will always remain and incorrect methods will sometimes be chosen. The RAP system copes with these problems by checking to make sure that each method achieves its intended goal and, if it doesn't, choosing another method and trying again.

The role of the RAP system within the overall architecture can be viewed in two different ways. On the one hand, the RAP system is a reactive planning system that can accomplish quite complex goals given appropriate methods in it's library and enough time to muddle through. On the other hand, the RAP system supplies the planner with abstract "primitive actions". Once a task (*i.e.*, a plan step) is selected for execution, the RAP system will work on it tenaciously, trying different methods and coping with whatever difficulties arise, until either the task is accomplished or every possible method has been tried. Thus, each task given to the RAP system, no matter how detailed or how vague, can be treated by the planner as a primitive action because the the RAP system attempts to guarantee success if the task is possible or fail predictably if not. The idea of RAPs as planning operators is discussed in more detail in [7].

3.1.1 Robust Action Control

One of the primary lessons of recent AI research into robot control is that actions must be active, dynamic processes tolerant of sensor error and changing surroundings: they must be reactive. It is not useful for a planning system to control a robot with steps like "move forward 2 feet" because the two feet may be hard to measure, the robot might slip sideways, an obstacle such as a person may suddenly appear, and so on. An action like "move forward 2 feet" really means move forward, avoid obstacles along the way, don't take too long, *etc.* Control systems must be given the flexibility to avoid undesirable states and must not be asked to execute actions with end states that cannot readily be measured.

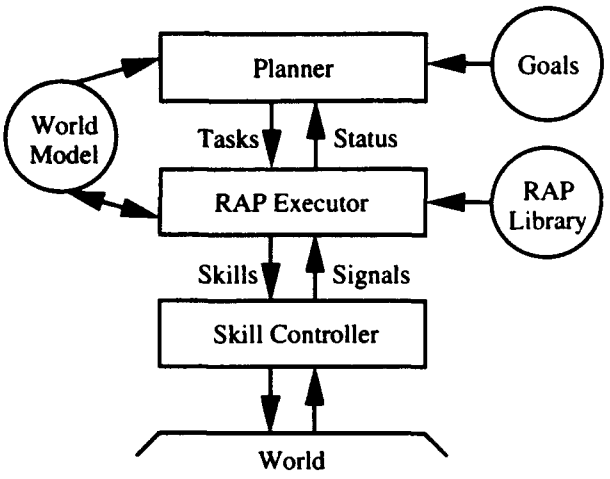


Figure 1: An Integrated Agent Architecture

Experience also shows that the most fruitful way to think of such a control system is as a collection of concurrent, independent behaviors. Each goal that the control system is designed to achieve becomes a family of feedback loops that work together to achieve the goal. Examples of casting the local navigation problem in this light can be found in [5] and [12].

3.2 Active Visual Routines

Strategies for gathering and using information about the world are just as important as those for taking action. Gathering information and determining the current situation for planning is often left to an agent's "perception system". However, we believe that perception should be treated just like action and that information gathering actions should be included with effector actions in plans and execution methods. Information gathering strategies thus become regular plans of action that include:

- Vision routines for extracting generally useful attributes of the world in reasonable time.
- Strategies for applying these techniques to produce robust descriptions of the relevant situation.
- Ways to combine new information with old in order to *recognize* objects and situations encountered in the past.
- Strategies for detecting and classifying the actions of another agent.

Once the situation has been determined, the executor will need strategies for defining and combining vision and behavioral routines into activities. Building activities is akin to constructing what are more traditionally called primitive actions and we believe that reactive skill construction can also be included in reactive plans incorporating:

- Visual techniques for continuous tracking of generally useful attributes of the world.
- Strategies for combining these techniques with actions to produce useful, robust goal-directed feedback loops.
- Ways to describe these strategies and techniques so they will transfer to new robots with new sensors and effectors.

Our goal in looking at both continuous activities and information gathering plans is to develop a general vocabulary for sensing and action that will serve a wide variety of everyday tasks. We believe that a vocabulary can be found that reflects the way the world works at a level that is independent of the details of any particular agent's set of sensors and effectors.

3.3 An Integrated Architecture

The Animate Agent Architecture attempts to integrate these ideas based on the idea of a *reactive skill*. A reactive skill corresponds to the notion of a discrete "primitive step" that can predictably, and reliably, be taken in the world. Execution of a skill also takes place over a period of time during which the control system is configured to carry out a specific action. The reactive execution system generates goal-directed behavior by refining the abstract plan steps of a sketchy plan into a sequence of appropriate, specific activities. It then implements each skill by generating a configuration for the control system.

Synthesizing control programs from scratch for arbitrary activities is an extremely difficult problem. Fortunately, it is not necessary. Drawing on ideas from behavioral control, we can divide a reactive skill into three components: the primary action required to achieve the skill's goal, an active sensing routine to continuously designate the "arguments" of the primary action in real time, and state monitoring processes to watch for and signal critical changes in the world, such as the achievement of a goal or the development of a problem that renders the current skill inappropriate. Each of these components can be implemented in advance as a modular routine that can be enabled and disabled by the reactive execution system.

The resulting interface between reactive execution and continuous control is illustrated in Figure 2. The RAP system takes tasks and refines them into appropriate activities based on the situation encountered at run time. Each skill is then further refined by choosing the appropriate primary action routine

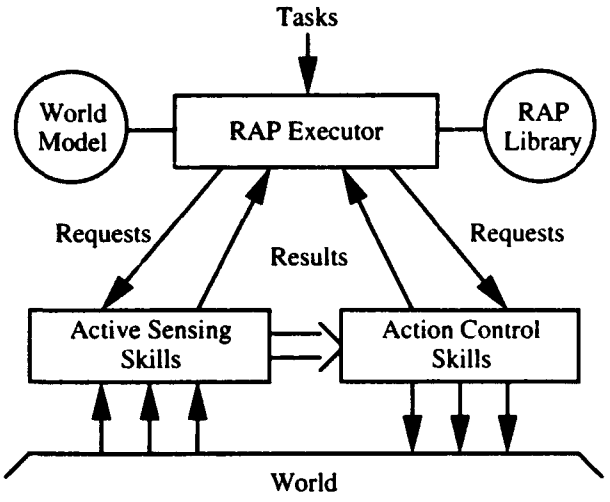


Figure 2: A Goal-Directed, Modular Control System

to implement its goal, an appropriate active sensing routine to track the object of the primary action, and state monitoring routines to ensure the detection of success or failure of the skill.

For example, suppose the next skill to be carried out is to move to a desk to vacuum underneath. This skill can be implemented using the fairly generic primary action routine "move in a given direction while avoiding obstacles." The argument to this routine is the direction to move and that can be updated in real time by an active sensing routine that continually tracks the direction to the desk. Possible routines might track the desk's shape, color, or motion, depending on the situation. It is up to the RAP system to choose which is appropriate at run time. Once the primary action routine and the tracking routine that supplies the direction to move have been enabled, the control system will move the robot towards the desk, whatever gets in the way. This control state, and reactive skill, will last indefinitely without further intervention as long as the tracked feature stays in view and the path toward it stays reasonably clear.

3.3.1 The RAP / Control System Interface

The interface between the RAP system and the modular control system is very simple. It consists of instructions to enable, disable, and set the parameters of individual routines. Thus, the RAP system simply configures the control system and cannot change the routines available or the attributes that connect them. Active tracking routines, when active, each compute the value for a fixed attribute, and primary action routines each require a fixed attribute. This restriction means that only certain combinations of routines make sense and the RAP system cannot change them. It also makes it easier to predict what effects a given, sensible, set of routines will have on the world.

Control routines communicate with the RAP system by sending signals. Signals are typically low information content messages such as "I've reached the position I was supposed to get to" or "I haven't made any progress in a while." Each routine will have a set of signals it sends under various circumstances. Since routines do not typically know the reason they have been enabled (*i.e.*, the goal enabling the routine is known to the RAP system but not the control system), the messages they send carry very little information when taken out of context. It is up to the RAP system to interpret these messages in light of the goals it is pursuing and the routines it has enabled. For example, "I cannot move forward", might mean failure because there is an obstacle in the way if the robot is trying to cross a room, on the other hand, the same signal might mean success if the robot is supposed to move down a long hallway until it reaches the end. The task expansion structure built up by the RAP system during execution is ideal for this interpretation.

The RAP system must be extended to allow it to enable multiple routines while maintaining the task expansion structure in which to interpret signals. Original work on the RAP system assumed discrete, symbolic actions that would each finish before the next needed to be chosen. However, since RAPs will now have to issues commands to enable a primary action, an active tracker, and some number of state monitors, the system will have to be able to continue past one enable to the next while keeping track of which goal spawned which routines. At the same time, RAP expansion will have to stop and wait whenever a control routine resource conflict arises.

3.3.2 A Modular Control System

The architecture assumes that activities can be implemented by enabling sets of routines. Each set would consist of: a primary action routine, an active tracking routine, and state monitor routines to watch for success, failure, and opportunities. A somewhat similar approach to configuring control systems was suggested in [10] but at a lower level of abstraction. The level at which control routines are defined and the resulting characteristics of those routines is very heavily influenced by the need to be aware of and respond to a constantly changing environment.

Primary Action Routines

Primary action routines correspond to control modes designed to accomplish a particular task, such

as moving in a given direction while avoiding obstacles, moving the arm to a particular location, or grasping a particular type of object.[1] Two strong constraints on algorithms implementing primary actions are: they must be able to respond in real-time to a changing workspace, and they must be able to act on moving, or changing, objects.

Our notion of primary action is analogous to the concept of guarded, compliant move used in the control of robotic arms. There, the controller uses force feedback to move the arm through some motion while continually adjusting applied forces until some end condition is reached. By selecting the correct compliance strategy for the initial conditions and desired behavior, quite complex tasks can be achieved robustly, even in the face of sensor and actuator uncertainty. We believe more generally that all primary actions must be "compliant" in nature, adapting in real-time to changing situations and assuming always that sensor and actuator response will be uncertain. This principle is a strong constraint on all algorithms we will be using for primary actions.

We also believe that primary action routines must be targetable in real time. That is, they must be constructed to act on some attribute of the world that can be sensed in real time rather than on an initial model. For example, moving in a given direction while avoiding obstacles is an important routine that can be used for a wide variety of different tasks like moving to a given location or approaching a given object. However, if the direction to move can be changed in real-time, the same routine can also be used to follow a moving person, or approach a moving object. Constructing action routines to act on changing targets is an important concession to the fact that the world is always moving.

Active Tracking Routines

Active tracking routines are the processes that compute attributes of the world for use in targeting primary actions. For example, the direction to an object can be used by routines designed to approach the object, to avoid the object, and to point in the object's direction. It is this direction attribute that has a functional use in many routines, not the object itself or the way that the attribute is extracted.

[1] Behavioral control systems are usually made up of much simpler routines with their interactions resulting in emergent properties that achieve specific tasks. However, the interdependencies between such simple routines are often so complex that the routines only make sense when activated together; arbitrary mixing and matching will typically lead to unpredictable behavior. It makes more sense to talk about reconfiguring a control system at the level of the primary task to be accomplished.

Agre and Chapman describe attributes of the world, like the direction to an object, as functional-indexical references [1]. The RAP system's choice of an active tracking routine to extract an attribute from the world is precisely the generation of a functional-indexical designation for that attribute. The sensing routine operationalizes exactly what is needed to reliably refer to and use an abstract symbolic property of the world. A desk being approached becomes "the-direction-I-am-tracking" and the approach skill uses "the-direction-I-am-tracking" as its reference. Together an active tracking routine and a primary action routine form a control loop to carry out a specific task.

Often there will be many different active tracking routines for extracting the same functional attribute from the world. This allows the RAP system substantial leeway to instantiate activities in different ways under different circumstances. For example, tracking the location of an obstacle might be done using color, shape, or motion, depending on what is most reliable in the specific context [14, 15]. When selecting a tracking strategy, the RAP system uses its knowledge of the obstacle to focus the interpretation of visual data (*i.e.*, color, motion, shape) down to just that which is necessary to reliably keep track of the object's relevant functional attribute (*i.e.*, position). Even when the desired skill is to move to a given spatial location, the general action for moving in a given direction can be used if it is coupled with a tracking routine that continually computes the current direction to that location based on dead-reckoning. Mixing and matching actions with target designators allows the same small number of routines to accomplish a wide variety of different goals.

Monitor Routines

Pairing a primary action routine with an active tracking routine is not enough to define a reliable task-directed skill. Each action must have a well-defined beginning and end. Moving forward while avoiding obstacles might have the effect of moving the robot down a hall, but when is that action complete? Completion may mean many things: having traveled a specific distance, having reached the end of the hall, or having passed three doorways. Regardless of which of these goals is desired, noticing that it has been achieved requires monitoring states in the world beyond those needed just to carry it out. These additional monitoring activities must be included in any skill designed to implement a discrete action.

Activities that define actions must not just signal completion; they must also reliably signal when they fail. The real world is filled with uncertainty and real sensors are not 100% accurate. As a result, a

primary action routine will not always achieve the goal it was intended to achieve. Therefore, the RAP system must augment the routines implementing the action with additional routines to monitor for failure states as well as for success states. For example, the robot might get stuck behind a step it cannot cross while approaching the desk. The tracker will continue to indicate the direction to the desk and the navigation routine will keep trying to move around the step but the robot won't be going anywhere. Another routine is required to watch that progress is being made. An skill must *always* signal that it is wedged. Only with such assurances can the planner and RAP system trust that their planned actions will interact with the world in a sensible fashion.

It should be noted here that we are not attempting to do "optimal" control. We are concentrating on building a system with the flexibility to carry out a wide variety of tasks and cope with mistakes and failures. We will never be able to guarantee success or optimality because of uncertainty and rapid change. The thrust of our project is to demonstrate the use of a variety of routines to perform many different tasks in a changing world. In other words, to show how to construct strategic plans for vacuuming that allow the actual vacuuming actions to execute reactively.

4 Conclusions

Autonomous vacuum cleaning is a very interesting task because it encompasses a variety of different tasks that progress from reactive, to synthetic, to intelligent. Each task requires a different architecture yet that architecture must continue to embody key ideas from its simpler cousins. At the low end of vacuuming complexity, a reactive architecture capable of dealing with misplaced furniture and the occasional moving object seems like the simplest solution to consider.

As the need to classify objects and vacuum around them in different ways becomes important, the architecture must change to one that represents explicit object classes and corresponding vacuuming plans. This change is required because the nature of the solution shifts from one naturally phrased in terms of concurrent processes to on naturally phrased in terms of goals and plans. However, while much of the knowledge and control structure within this new synthetic architecture will be symbolic, actual actions taken in the world must remain reactive to cope with the same uncertainties and changes the simple architecture handles well.

Finally, as the need to interact with human users becomes important, the architecture must further evolve to include a clear understanding of the "greater environment," the robot's own goals and plans, and the apparent goals and plans of the user. Such an intelligent vacuum cleaner must continue to deal in symbols and reactive processes even as it attempts to recognize and account for the needs and desires of other agents.

4.1 The Animate Agent Project

The Animate Agent project at the University of Chicago is attempting to address these issues by describing and building a synthetic architecture that combines a symbolic, reactive planner (the RAP system) with a modular, control system consisting of a library of continuous sensing and action processes. The RAP system is good for choosing different courses of action in different contexts, maintaining a reasonably up-to-date view of the current situation, and supplying the context for interpreting the results of sensing and action operations. The modular control routines are a natural way for describing the continuous, low-level actions required for actually interacting with the world. The RAP system carries out its goals by refining them down into activations and deactivations of control (and sensing) routines.

We believe that this architecture and the vocabulary of low-level control routines we are building to go with it are perfectly suited to the vacuum cleaner domain. The test will be to implement the plans and routines needed to carry out a variety of vacuuming strategies in a number of different situations.

References

[1] Philip E. Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Sixth National Conference on Artificial Intelligence*. Seattle, WA, July 1987. AAAI.

[2] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1), March 1986.

[3] Thomas L. Dean and Michael P. Wellman *Planning and Control*. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1991.

[4] R. James Firby. Adaptive execution in complex dynamic worlds. Technical Report YALEU/CSD/RR #672, Computer Science Department, Yale University, January 1989.

[5] Erann Gat. *Reliable Goal-Directed Reactive Control of Autonomous Mobile Robots*. PhD thesis, Computer Science and Applications, Virginia Polytechnic Institute, 1991.

[6] Michael P. Georgeff, Amy L. Lansky, and Marcel J. Schoppers. Reasoning and planning in dynamic domains: An experiment with a mobile robot. Tech Note 380, AI Center, SRI International, 1986.

[7] Steven John Hanks. Projecting plans for uncertain worlds. Technical Report YALEU/CSD/RR #756, Computer Science Department, Yale University, January 1990.

[8] Leslie Pack Kaelbling. Goals as parallel program specifications. In *Seventh National Conference on Artificial Intelligence*, St. Paul, MN, August 1988. AAAI.

[9] Frank L. Lewis. *Optimal Control.* John Wiley and Sons, New York, New York, 1986.

[10] D.W. Payton. An architecture for reflexive autonomous vehicle control. In *International Conference on Robotics and Automation*, San Francisco, CA, 1986. IEEE.

[11] M.J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987. IJCAI.

[12] Marc G. Slack. Situationally driven local navigation for mobile robots. Technical Report JPL Publication 90-17, Jet Propulsion Laboratory, April 1990.

[13] M.G. Slack. Sequencing formally defined reactions for robotic activity: Integrating raps and gapps. In *Vol. 1828 Sensor Fusion V: Simple Sensing for Complex Action*, Boston, MA, November 1992. SPIE.

[14] Michael J. Swain. Color indexing. Technical Report 360, Department of Computer Science, University of Rochester, 1990.

[15] Michael J. Swain and Dana H. Ballard. Color indexing. *International Journal of Computer Vision*, 7:11–32, 1991.